AVICA: An Access-time Variation Insensitive L1 Cache Architecture

Seokin Hong and Soontae Kim
Department of Computer Science
Korea Advanced Institute of Science and Technology
{seokin, kims}@kaist.ac.kr

***Abstract*—Ever scaling process technology increases variations in transistors. The process variations cause large fluctuations in the access times of SRAM cells. Caches made of those SRAM cells cannot be accessed within the target clock cycle time, which reduces yield of processors. To combat these access time failures in caches, many schemes have been proposed, which are, however, limited in their coverage and do not scale well at high failure rates. We propose a new L1 cache architecture (AVICA) employing asymmetric pipelining and pseudo multi-banking. Asymmetric pipelining eliminates all access time failures in L1 caches. Pseudo multi-banking minimizes the performance impact of asymmetric pipelining. For further performance improvement, architectural techniques are proposed. Our experimental results show that our proposed L1 cache architecture incurs less than 1% performance hit compared to the conventional cache architecture with no access time failure. Our proposed architecture is not sensitive to access time failure rates and has low overheads compared to the previously proposed competitive schemes.**

I. Introduction

Performance, density and energy efficiency of microprocessors have been improved following Moores law for past decades. However, as we approach a nanotechnology era, we are facing reliability problems. Process variations, mainly due to random-dopant fluctuations (RDF) make microprocessors more unreliable by varying their latencies, and by increasing their power consumption [1]. SRAM structures are usually implemented with minimum feature size transistors to reduce their area and power consumption, which makes them more vulnerable to process variations [2].

Due to process variation in SRAM cells, caches made of those SRAM cells face three kinds of failure: read failure, write failure, and access time failure. Among them, the access time failures are the most dominant, especially in severe process variation environments as shown in Fig. 1. An access time failure occurs when the cell access time, which is defined as the time needed to discharge the bitline of a cell for a read operation, exceeds the maximum tolerable limit.

Several techniques have been proposed to address the access time failures in caches. Variable latency cache architecture is studied in [3], [4]. A wordline boosting technique is proposed in [5]. Error Correcting Code (ECC) and disabling/redundancy based techniques [6], [7], [8], [9] can be used to tolerate the access time failures as well as the read and write failures. All of these techniques, however, are limited by their coverage at high failure rates.

978-3-9815370-0-0/DATE13/

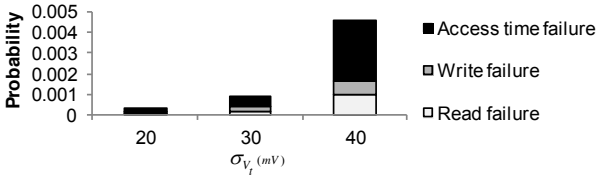


Fig. 1. The failure probability of 6T-SRAM cell [2]

In this paper, we propose a new L1 cache architecture, called **AVICA**, which is insensitive to the access time failure rates. AVICA employs two design concepts: **asymmetric pipelining** and **pseudo multi-banking**. Asymmetric pipelining makes the L1 caches free of access time failures by allowing cache pipeline stages to have asymmetric cycles; the cell access stage takes two cycles for eliminating access time failures, while the other stages take one cycle. Pseudo multi-banking gets back the cache bandwidth loss due to asymmetric pipelining. Unlike conventional multi-banking, pseudo multi-banking accepts a single memory access per cycle while it allows multiple memory accesses to different banks at the same time, which greatly simplifies our multi-bank design. In addition, we add small buffers to store recently accessed data. On hits in the buffers, we can avoid bank conflicts and obtain requested data earlier. Conventional issue queue is modified to utilize this shorter access latency.

Our experimental results show that AVICA with the optimal bank configurations incurs performance penalties of less than 3% on average, compared to conventional cache architecture with no access time failures. With additional buffers, the performance hit reduces to less than 1% on average.

The rest of this paper is organized as follows. We discuss related work in the next section. Section 3 discusses the motivation of our approach. Section 4 explains our proposed cache architecture. Section 5 introduces additional architectural supports for further improvements. Experimental results based on cycle-accurate simulations are given in section 6. Section 7 concludes this paper.

II. Related Work

SRAM cell design: The impact of variations can be mitigated using wider transistors. Doubling transistor channel area enables 30% reduction in Vt variations [2]. However, wider transistors consume more static power and more dynamic energy in switching channels. Novel topologies like 8T-, 10T-, and ST SRAM cell are proposed to mitigate stability failure.

TABLE I
COMPARISON OF COMPETITIVE SCHEMES

	AVICA	VL-cache [3]	Wordline boosting [5]	DECTED	ZC [7]
Area overhead(%)	4	21	4.5	68	16
Performance overhead(%)	1	3.6	1	1	1
Power overhead(%)	3	9.7	9.8	51	16
Target cell failure rate	$\gg 0.003$	0.003	0.003	0.001	0.001
Addressable failure model	access time	access time	access time	all	all

All these topologies increase the array complexity and the area cost. Moreover, they cannot tolerate access time failures which are the problem addressed in this paper.

Error correction code: The SECDED (Single-error correction, double-error detection) is widely used to protect memory against transient errors. However, they are not practical in tolerating high failure rates. While multi-bit error correction codes (e.g. DECTED, and QECPED) can tolerate high failure rates, they are not applicable to L1 caches due to significant area, latency and power overheads [10]. Kim et al. proposed 2D error coding to tolerate multi-bit errors with low area overhead [10]. However, it has large correction latency and thus is not applicable to L1 caches.

Disabling and Redundancy: Disabling and redundancy based techniques are proposed to enhance stability, especially at low voltage mode. Agarwal et al. [6] proposed a block remapping technique. Faulty blocks are remapped to the neighboring functional blocks. Wilkerson et al. [8] proposed a word-level disabling technique. Two consecutive faulty blocks in a cache set can be combined to form a functional block. ZerehCache [7] provides a spare line to a group of cache lines, such that each word in the spare line can substitute one faulty word in the group. These techniques require considerable amount of extra memory structures or complex interconnection network.

Latency-aware architecture: Yan Pan et al. proposed a selective wordline voltage boosting. By boosting the voltage on wordlines of slow cache lines, access time failures can be tolerated [5]. This technique is effective under medium level variations. At severe variations, it is required to boost the voltage on the most wordlines. In addition, It has limited capability of speeding up slow SRAM cell access; SRAM cell access latency is reduced by 18% as the wordline voltage is increased from 0.9V to 1.3V. Moreover, the increase in wordline voltage reduces the lifetime of the memory cell [3]. Variable latency cache (VL-cache) architectures are proposed in [3], [4]. In these techniques, the access latency of each cache line is stored in a buffer. Depending on the latency information, it accesses the cache lines at different speeds. The slow cache access reduces the cache bandwidth and increases the hit latency. Thus, the performance overhead depends on the frequency of slow cache access. At severe variations, it is expected to access all cache lines slowly, which results in considerable performance loss.

Summary: As discussed above, none of the previously proposed schemes can handle access time failures effectively and efficiently at severe variations as summarized in Table I.

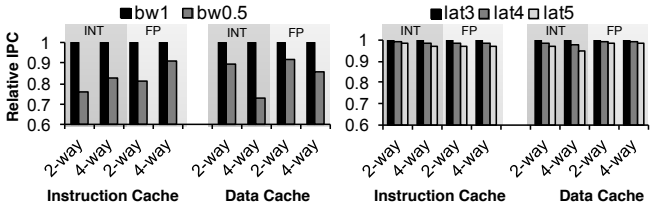


Fig. 2. Performance impacts of bandwidth reduction (left) and latency increase (right); "bw1" indicates that the bandwidth is one word and one fetch group per a cycle for data and instruction caches, respectively, while "lat3" indicates hit latency of three cycles

III. MOTIVATION: HIT LATENCY TOLERANCE

Our motivation comes from an important observation about the performance impact of the hit latency and the bandwidth of L1 caches. Due to the low latency and the high bandwidth of L1 caches, the processor can minimize memory access time and enhance performance. In case of the out-of-order superscalar processors, however, a small increase in the hit latency of L1 caches slightly affects performance. This is because out-of-order execution and accurate branch prediction provide **hit latency tolerance** [11], [12]. The out-of-order execution allows ready instructions, which are independent from a long-latency load instruction, to be executed without stalls. Thus, even if the hit latency of L1 data cache increases by one or two cycles, its performance impact is small. The L1 instruction cache is typically pipelined so that its long hit latency can be overlapped if branch predictions are accurate. Since the branch prediction accuracies of modern processors are very high, more than 96% for SPEC 2006 benchmarks in the intel i7 [13], the performance is not much affected by the increase in the hit latency of L1 instruction cache.

We studied the performance impact of the hit latency and bandwidth of L1 caches in two architecture configurations: 4-way and 2-way out-of-order processors. The detailed experimental environments are presented in section 6. Fig. 2 shows normalized average IPC for each configuration. Firstly, we vary cache bandwidth while the hit latency is fixed at 3 cycles. Performance loss is significant when bandwidth is reduced to half. The reduced instruction cache bandwidth results in more than 18% and 11% performance losses in 2-way and 4-way processors, respectively. The 4-way processor is much more affected by data cache bandwidth reduction. It suffers from almost 30% performance loss for the integer benchmarks with reduced bandwidth. Next, we vary the hit latency of caches from 3 to 5 cycles while bandwidth is fixed at one word per cycle for data cache (one fetch group per cycle for instruction cache). As the hit latency increases, both 2-way and 4-way processor experience slight performance loss. Even with two-cycle increase in the hit latency, the performance loss is less than 6% for both processors. One cycle increase in the hit latency results in less than 1.2% performance loss.

The hit latency tolerance motivates us to design a novel cache architecture to combat against cache access time variations. **We increase hit latencies of the L1 caches to eliminate cache access time failures without sacrificing cache access bandwidth by engineering the L1 caches.**

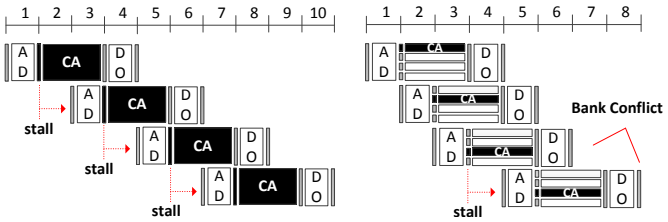


Fig. 3. Operational example of asymmetric pipelining (left) and asymmetric pipelining with pseudo multi-banking (right)

IV. Proposed Access time Variation Insensitive L1 Cache Architecture

Our proposed cache architecture (AVICA) employs two design concepts: asymmetric pipelining and pseudo multi-banking. Asymmetric pipelining eliminates all access time failures. However, it reduces the effective cache bandwidth. Pseudo multi-banking makes it possible to get back the lost cache bandwidth.

A. Eliminating access time failures with Asymmetric Pipelining

Cache accesses require multiple cycles in modern high-performance processors [14]. This is mainly because bitline and wordline delays of caches do not scale well with new technology generations [14]. Since multi-cycle cache accesses significantly reduce cache bandwidth and, consequently, overall performance, caches are typically pipelined. In this paper, we assume that the L1 caches have three pipeline stages; address decode (AD), cell access (CA) and data out (DO).

Due to the within-die variation, each pipeline stage of caches can be faster or slower than the design target. In order to take into account this problem, we propose an **asymmetric pipelining**. Unlike the conventional cache pipelining, the access latency of CA stage is longer than the other stages in the asymmetric pipelining as shown in Fig. 3 (left) (i.e., two clock cycles for CA stage while one clock cycle for the other stages). Since we give enough timing margin to CA stage, access time failures do not occur assuming the slowest cell access takes two cycles. In other words, our asymmetrically pipelined caches are not affected by access time failure rates. AD and DO stages also experience latency variations. However, they are less vulnerable to process variation because there is a large number of transistors on a critical path in these stages; positive and negative variations in speed of the transistors cancel each other. Moreover, the total number of transistors in these stages is much smaller than that of CA stage. Thus, upsizing transistors efficiently tolerates latency variation in these stages.

The access latency of CA stage in the asymmetrically pipelined cache is two cycles and it is not pipelined into two stages. Since two cache accesses cannot reside in CA stage at the same time, one stall cycle is inserted after a cache access is initiated, which reduces the effective cache bandwidth.

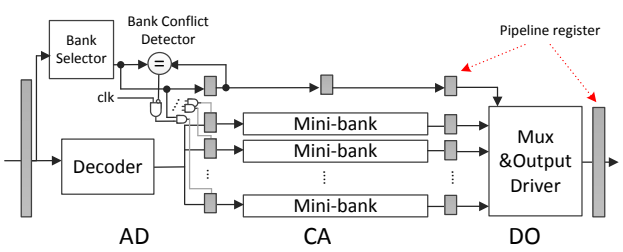
B. Restoring bandwidth through Pseudo Multi-banking

Asymmetric pipelining decreases cache bandwidth, which can result in significant performance degradation. In order to overcome this limitation of asymmetric pipelining, we employ a low cost multi-banking technique called **pseudo multi-banking**. Cache banking has been widely used for designing high bandwidth (multi-banking) [15] or power-optimized (sub-banking) [16] caches. In the high bandwidth caches, multiple memory accesses are accepted and distributed to multiple banks while a single bank (or sub-bank) is activated to service a single memory access in the power-optimized caches. Note that the primary objective of employing pseudo multi-banking is to restore bandwidth, not to enhance bandwidth. Thus, unlike conventional multi-banking, pseudo multi-banking accepts a single memory access per cycle. But, it allows multiple memory accesses to reside in different banks at the same time, which makes it possible to overlap two long-latency cell accesses. Consequently, as shown in Fig. 3 (right), CA stage (takes two cycles) can have two accesses at the same time if they go to different banks, which restores the bandwidth loss. If the two consecutive memory accesses go to the same bank, then one stall cycle is still required due to a bank conflict.

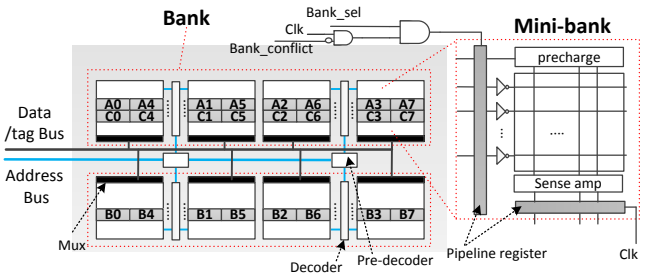
C. Implementation

An overall cache architecture with asymmetric pipelining and pseudo multi-banking, shown in Fig. 4 (a), is not much different from the conventional cache architecture, which results in low additional implementation cost. For address decoding, a single address decoder is employed because at most one address decoding is required in a cycle. The complex crossbar networks in input and output ports, which is the most expensive components in the conventional multi-banked cache [15], are not required. In addition, a cell array of the conventional cache is typically partitioned into multiple subarrays or subbanks to reduce the cell access time and power consumption [14], [16]. Only differences are that we need to select a bank to be accessed, detect bank conflicts, and control wordline and bitline precharging circuit of each bank independently. To this end, we employ three additional components: a bank selector, a comparator and a few AND gates. A bank selector determines a bank number from a memory address. In order to detect bank conflicts, we compare the previous bank number, stored in a pipeline register, with the bank number of current memory access. If a bank conflict is detected, we prevent the current memory access from traversing down the pipeline stages. To this end, the clock signal is ANDed with the bank select signal and bank conflict signal, which allows only the pipeline register of the selected bank to latch new decoded wordline value and bitline precharging signal when bank conflicts do not happen.

Two dimensional interleaving: Due to bank conflicts, the bandwidth gain of pseudo multi-banking will be reduced. In order to minimize the bank conflicts in pseudo multi-banking, we exploit line-interleaving and word-interleaving simultaneously. The data array is horizontally divided into banks and each bank is vertically subdivided into small sub-arrays, called **mini-banks**. Each mini-bank can be activated independently so that only a single mini-bank containing the desired data can be accessed at a time. We place consecutive



(a) Overall architecture



(b) Layout of an example array with 2 banks. In this example, each bank has 4 mini-bank

Fig. 4. Implementation of AVICA

cache lines in different banks and place consecutive words of each cache line in different mini-banks. Through two dimensional interleaving, we can efficiently reduce bank conflicts because memory accesses typically show spatial locality. Note that word-interleaving is costly in the conventional multi-banked cache due to the need for tag replication in each bank to allow simultaneous cache accesses [15]. In contrast, the cost for word-interleaving is not required in pseudo multi-banked cache because it accepts a single memory access per cycle.

Fig. 4(b) illustrates two dimensional interleaving with the layout of an example array. The data array is divided into two banks and each bank is subdivided into four mini-banks. In this example, we assume that three cache lines (A, B and C) are consecutive. Thus, A and B are stored in different banks, and B and C are stored in different banks. Each cache line consists of eight words (e.g. A0 $\sim$ A7 for line A) being interleaved among multiple mini-banks.

The tag array is not banked but it is implemented with upsized transistors. Since the tag array consumes much lower area than the data array (around 3$\sim$4% of a 32KB 4-way set-associative cache), upsizing the transistors of the tag array incurs low area and power overheads.

V. Architectural Additions for Further Improvement

Through asymmetric pipelining and pseudo multi-banking, AVICA becomes free of the process variation-induced access time failures. However, some benchmarks, especially integer benchmarks, show nontrivial performance hits as will be shown in section 6. Thus, there is still a room for improvements in those benchmarks. To this end, we present several architectural additions in this section.

A. Minimizing bank conflicts

Branch target instruction buffer: In AVICA I-cache (L1 instruction cache), sequential instruction flow does not incur bank conflicts due to word-level interleaving. However, when a taken branch changes the instruction fetch flow, the current and next fetch groups may happen to be in the same mini-bank, which leads to bank conflicts. In order to minimize bank conflicts in AVICA I-cache, we employ a branch target instruction buffer (BTIB), which is illustrated in Fig. 5(a). BTIB is a tiny fully associative cache which stores recently accessed instructions incurring bank conflicts and located at branch targets. Each entry of BTIB contains two fields: fetch address (ADDR) and branch target instructions (BTI). Branch target address is stored in the ADDR field and the target instructions (four instructions for 4-way superscalar processors) are stored in the BTI field. BTIB is accessed at AD stage of the cache pipeline. When a hit in BTIB occurs, instructions stored in the BTI field are sent to the instruction fetch queue while CA stage of the cache pipeline is skipped. Thus, the current fetch request does not incur a bank conflict. Consequently, next fetch request is also free of a bank conflict. Branch target instructions can be stored in BTB rather than using BTIB [13]. However, it is not efficient in terms of area and power consumption because the performance hit of AVICA I-cache becomes ignorable with a small number of BTIB entries as will be shown in section 6.

Word buffer: For AVICA D-cache (the L1 data cache), consecutive memory references to the same word (e.g. load byte) go to the same mini-bank, which leads to bank conflicts. We denote this kind of bank conflict as a same word conflict. Note that applications frequently use sub-word data types (e.g. the size of "int" data type is still 32-bit wide in many 64-bit complier including Solaris, Linux, FreeBSD, etc.). To overcome this limitation, we employ a word buffer (WB). WB stores recently accessed data words. The AVICA D-cache and WB are accessed simultaneously. If a load hits in WB, the requested data is obtained from WB, not from the AVICA D-cache. Thus, the current cache access does not incur a bank conflict and the next cache access is also free of a bank conflict. On a miss in WB, the data obtained from the AVICA D-cache is stored in WB. The store instructions write data to both the AVICA D-cache and WB to keep coherence between them.

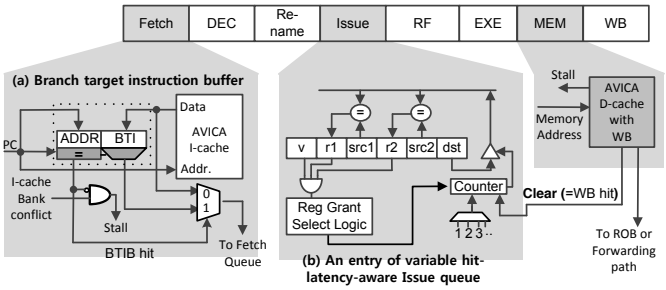


Fig. 5. Architecture pipeline with Branch Target Instruction Buffer (BTIB), Word Buffer (WB) and Variable Hit-latency-aware Issue Queue (VHIQ); Structure of WB is identical with that of BTIB

B. Compensating increase in hit latency

Variable hit latency-aware issue queue: Even though the out-of-order processors generally show hit latency tolerance, some benchmarks, especially integer benchmarks, experience noticeable performance degradation in AVICA D-cache. To alleviate this limitation, we introduce variable hit latency-aware issue queue (VHIQ) which cooperates with WB. WB consists of a small number of entries and implemented with upsized transistors. Thus, it is faster than AVICA D-cache and is hardly affected by process variation; we assume that the access time of WB is three cycles while that of the AVICA D-cache is four cycles. If a memory reference hits in WB, the requested data can be obtained from WB in three cycles, not four cycles. Meanwhile, in out-of-order superscalar architecture, dependent instructions on loads are issued speculatively assuming the loads will have specified fixed latency [17]. Thus, in order to effectively exploit the fast accessible WB, we need to notify the issue queue that a load instruction hits in WB so that dependent instructions on the load can be issued earlier.

Fig. 5(b) shows one entry of a conventional issue queue [18]. When a load is issued, it sets the counter to four, which is the hit latency of AVICA D-cache. Four cycles later, the counter will expire and wake up the dependent instructions. Therefore, issuing the dependent instructions can be advanced by clearing the counter value. When a load instruction hits in WB, the clear signal of its issue queue entry is activated. After clearing counter value, the destination field of the load instruction's issue queue entry is broadcasted on the result tag bus to wake up dependent instructions. Modification to the conventional issue queue is minimal.

VI. Evaluation

A. Experimental Environment

We carry out architectural experiments using an execution-driven simulator MASE [19]. Detailed architecture parameters are given in Table II; In this paper, the baseline architecture is 4-way superscalar out-of-order processor. We use seven floating point and ten integer benchmarks from SPEC CPU2006 suit. For all benchmarks, we fast-forward 10 billion instructions and perform cycle-accurate simulation for next 1 billion instructions. To evaluate the power and latency of AVICA caches and additional buffers, we use CACTI 6.5.

TABLE II
Architectural configuration of simulated processor

Parameter	Configuration	
	4-way out-of-order	2-way out-of-order
Fetch/issue/commit	4/4/4	2/2/2
Window size	Issue queue (32), ROB (96), LSQ (32)	Issue queue (16), ROB (48), LSQ (16)
Function units	Int Add /Mult (4/1), FP Add/Mult (4/1)	Int Add/Mult (2/1), FP Add/Mult (2/1)
L1 I-cache/D-cache	32KB, 4-way, 64B line, 3 cycles **(4 cycles for asymmetric pipelining)**	32KB, 4-way, 32B line, 3 cycles **(4 cycles for asymmetric pipelining)**
Unified L2 cache	2MB, 8-way, 64B line, 12 cycles	512KB, 8-way, 32 B line, 12 cycles
Memory	150 cycles for the first chunk	80 cycles for the first chunk
BTB	2048 entry, 4-way	
Branch Predictor	comb. of bimodal and 2-level global	
TLB	ITLB (16-set, 4-way), DTLB (32-set, 4-way), 4KB page size, 30 cycle penalty	

B. Performance results

Fig. 6 compares the performance impacts of asymmetric pipelining with several configurations. Each bar in the figure shows the performance degradation with respect to the baseline cache architecture (hit latency is three cycles). For the instruction cache, without pseudo multi-banking, asymmetric pipelining incurs significant performance degradation (more than 10% on average) as shown in Fig. 6 (top). With perfect banking (no bank conflict), however, its performance loss becomes lower than 1.1% on average. Through pseudo multi-banking with the (2,8) bank configuration (2 bank, 8 mini-bank), which is an optimal configuration for AVICA I-cache, performance loss becomes lower than 1.9% and 1.2%, on average, for the integer and floating point benchmarks, respectively, which is almost comparable to that of perfect banking.

The results for the data cache show similar behaviors to those for the instruction cache as shown in Fig. 6 (bottom). With (4,4) bank configuration, which is an optimal configuration for AVICA D-cache, it experiences average 3% and 1.4% performance degradation for the integer and floating point benchmarks, respectively, while it shows 20% performance degradation, on average, without pseudo multi-banking. The integer benchmarks are generally more sensitive to the bandwidth loss and increase in hit latency than the floating point benchmarks. The bzip2, gobmk, and sjeng benchmarks experience more than 3.6% performance degradation even

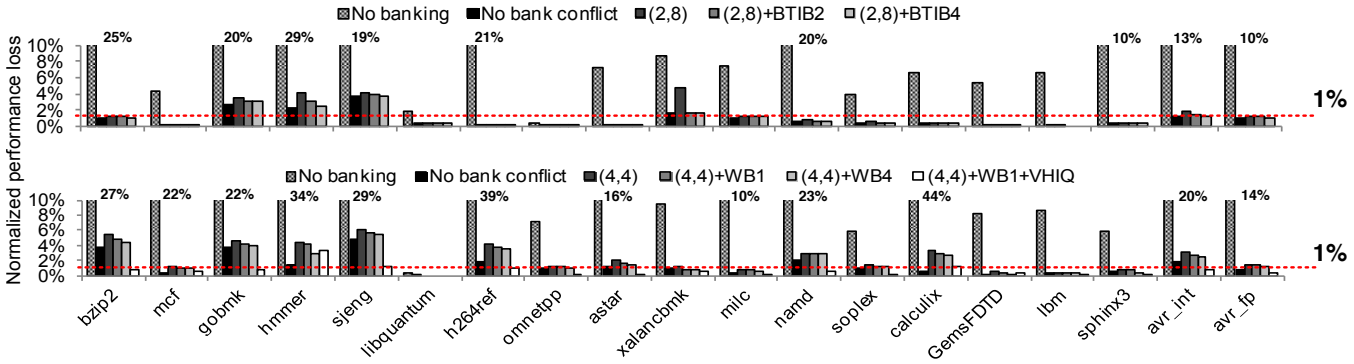


Fig. 6. Performance overhead; instruction cache (top) and data cache (bottom); a legend (a,b): a - the number of bank, b - the number of mini-bank

Fig. 7. Area overhead of the competitive techniques with and without AVICA for tolerating a cell failure rate of 0.001

with perfect banking. This is because these benchmarks show relatively high IPCs and have a large number of instructions which depend on the load instructions (data dependency).

For AVICA I-cache, BTIB nullifies the performance overheads incurred by bank conflicts, especially for the banking-unfriendly benchmarks such as hmmer and xalancbmk. With four-entry BTIB (BTIB4), performance impact of pseudo multi-banking approaches that of perfect banking across almost benchmarks. The performance overhead becomes less than 1% on average for both the integer and floating-point benchmarks.

For AVICA D-cache, even if WB noticeably reduces performance loss for some benchmarks such as bzip2 and h264ref, it is not so effective when used alone. Cooperation of WB and VHIQ significantly improves performance; the performance degradation becomes less than average 1% for both the integer and floating point benchmarks with one entry-WB (WB1).

C. Implementation costs and limitations

As we discussed in section 4, AVICA add a small number of additional costs to the conventional L1 cache architecture. Main additional components are some circuitry for detecting bank conflicts and controlling pipeline registers of cache banks independently. Their area and power costs are very small. We employ additional buffers: BTIB and WB. As we have seen above, performance hits become ignorable with a small number of entries for these buffers. The area overhead of the additional buffers is less than 0.2% of the L1 caches. Major overhead of our architecture comes from upsized transistors of the tag array. Area and power overheads are less than 4% and 3%, respectively, which are obtained by our CACTI simulation. In this evaluation we do not take into account the area and power overhead of AD and DO stages, which are protected by using upsized transistors as we discussed in section 4. We note that the impact of this overhead would be the same with all competitive techniques. The implementation overheads of AVICA and competitive techniques are summarized in Table I.

AVICA targets only access time failures, therefore it has to be used together with other schemes targeting the read and write failures. Since AVICA eliminates all the access time failures, which is most dominant, other schemes will be able to handle the read and write failures more effectively with lower cost as shown in Fig. 7.

VII. Conclusion

Variations due to imprecise process technology results in large deviations of SRAM cell access times in caches. To address this problem, we proposed a novel L1 cache architecture (AVICA) which is insensitive to access time failure rates. AVICA achieves its goal with two design concepts: asymmetric pipelining and pseudo multi-banking. Asymmetric pipelining eliminates all access time failures. Pseudo multi-banking gets back the access bandwidth loss resulting from asymmetric pipelining. In addition, we employ additional small buffers and extend the issue queue. Experimental results demonstrate that the performance impact of AVICA is less than 1% on average. **Most important feature of our proposed architecture is its insensitivity to access time failure rates, which will be more important in future processors with severe process variations**.

Acknowledgment

This research was supported by the National Research Foundation of Korea (NRF) grants funded by the Ministry of Education, Science and Technology (2011-0005378, 2012-0000980) and by the Ministry of Knowledge and Economics (10041313).

References

[1] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *Ieee Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[2] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 12, pp. 1859–1880.

[3] M. Mutyam, F. Wang, R. Krishnan, V. Narayanan, M. Kandemir, Y. Xie, and M. J. Irwin, "Process-Variation-Aware Adaptive Cache Architecture and Management," *IEEE Transactions on Computers*, vol. 58, no. 7, Jul. 2009.

[4] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou, "Yield-Aware Cache Architectures," in *MICRO 39*.

[5] Y. Pan, J. Kong, S. Ozdemir, G. Memik, and S. W. Chung, "Selective wordline voltage boosting for caches to manage yield under process variations," in *DAC '09*.

[6] A. Agarwal, B. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A process-tolerant cache architecture for improved yield in nanoscale technologies," *IEEE Transactions on VLSI Systems*, vol. 13, no. 1, pp. 27–38, 2005.

[7] A. Ansari, S. Gupta, S. Feng, and S. Mahlke, "ZerehCache: armoring cache architectures in high defect density technologies," in *MICRO 42*.

[8] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," in *ISCA '08*.

[9] T. Mahmood and S. Kim, "Fine-Grained Fault Tolerance for Process Variation-Aware Caches," in *ISVLSI '10*.

[10] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding," in *MICRO 40*.

[11] T. M. Austin and G. S. Sohi, "Zero-cycle loads: microarchitecture support for reducing load latency," in *MICRO 28*.

[12] S. T. Srinivasan and A. R. Lebeck, "Load latency tolerance in dynamically scheduled processors," in *MICRO 31*.

[13] J. Hennessy and D. Patternson, *Computer Architecture: A quantitative Approach*, 5th ed., Sep. 2011.

[14] A. Agarwal, K. Roy, and T. N. Vijaykumar, "Exploring High Bandwidth Pipelined Cache Architecture for Scaled Technology," in *DATE '03*.

[15] J. Rivers, G. Tyson, E. Davidson, and T. Austin, "On high-bandwidth data cache design for multi-issue processors," in *MICRO 30*.

[16] K. Ghose and M. B. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation," in *ISLPED '99*.

[17] R. Kessler, "The Alpha 21264 microprocessor," *Micro, IEEE*, vol. 19, no. 2, pp. 24–36, 1999.

[18] D. Ernst and T. Austin, "Efficient dynamic scheduling through tag elimination," in *ISCA '02*.

[19] E. Lason, S. Chatterjee, and T. Austin, "MASE: A Novel Infrastructure for Detailed Microarchitectural Modeling," in *ISPASS '01*.